Simulating Software Architectures for Functional Analysis

Henry Muccini and Patrizio Pelliccione
Dipartimento di Informatica
Università degli Studi dell’Aquila, Italy
{muccini,pellicci}@di.univaq.it

Abstract

Simulation is a mean for verifying the quality of an architectural specification. Some approaches have been proposed in the past. Each approach has its own internal simulation engine, and allows for certain types of simulations. Goal of this paper is to propose SASIM, a software architecture-level simulation technique for modeling and simulating architectural specifications. It improves existing techniques, by permitting various kinds of simulation, and making use of SPIN as external simulator. This allows a real and useful integration between simulation and exhaustive verification. The paper discusses SASIM and compares it with existing techniques.

1 Introduction

Software Architectures (SA) play a central role in the development of today complex and critical systems. They are utilized for documenting architectural design decisions and solutions, for assessing the architectural quality, for predicting the final system characteristics, and for guiding the design and coding process. Assuring as early as possible the correctness of an SA occupies an increasing importance in the development life-cycle.

Simulation is one of the most practiced methods for design validation. Simulation aims at exploring the effects of certain design decisions or environmental conditions on system behavior. It allows designers to better understand the functioning of complex elements of the real world, helping in producing better results and faster. Simulation is explorative, not exhaustive. It can be also successfully used in conjunction with exhaustive verification: the combination of simulation and verification has been proven to be an effective mean to increase state-space coverage or to restrict the search space to be explored via simulation [6].

In the domain of software architectures, simulation can help software architects exploring the architectural model prototype, so to detect architectural decision inconsistencies and to improve the architectural design model. The benefits are manifold: when different architectural design decisions are possible, pros and cons of each of them can be easily predicted by simulation; the impact of an architectural change can be immediately analyzed, thus understanding the implications (in terms of time, performance, or behavior) of the change on the entire system; simulation can help to make an estimate on components and tasks dependencies, delays, and resources.

Some approaches have been proposed in the last years to exploit the power and the effectiveness of simulation in the context of software architectures [1, 2, 5, 9]. They have been used both for quantitative and qualitative analysis.

This paper presents SASIM, a simulation approach that offers the following main advantages with respect to previous works: i) it covers different types of simulation; ii) it offers a way to integrate simulation and model-checking, thus reducing the search space to be explored by model checkers and improving the overall analysis cost and effectiveness; iii) it offers a solution that, by taking advantage of the SPIN [7] simulator and model checker engines, provides the architect with an architectural language front end, and offers a logic to transform simulation results from the SPIN level to the architectural model level.

2 SASIM

The idea of SASIM is to create a tool that exploits the SPIN potentialities, and in particular its simulator, hiding the complexity to the software architecture designer. As in any other simulation activity, two are the main phases: modeling and simulation. As it is shown in Figure 1, starting from a software architecture described in terms of CHARMY [8, 3] state, sequence, and topology diagrams a Promela code is generated. The Promela code is used by SPIN for both model checking the system and simulating it. The results of the verification and of the textual simulation outputs are translated into CHARMY sequence and state diagrams.

In SASIM, the software architecture is specified us-
The CHARMY editor. CHARMY is an architectural specification and verification environment which supports both model-based specification of an architecture, model-checking, and testing. CHARMY allows the specification of the SA topology in terms of components, connectors, and relationships among them. The internal behavior of each component is specified in terms of CHARMY state and sequence diagrams. The CHARMY notation for state machines permits to specify the intra-component and inter-component behaviors of architectural components and connectors. Sequence diagrams are used to specify how components communicate.

As far as concern simulation, SASIM makes use of the SPIN [7] model checker and its simulation features. While the use of an external simulator avoids the development of an ad-hoc simulator, the use of the SPIN simulation features does not come for free. Two, in fact, are the main tasks to be managed for realizing an architecture-based simulation based on SPIN: the SA specification needs to be translated into the input language of SPIN; simulation outputs need to be expressed in terms of the architectural models. In the following, more details are provided on both actions.

Once the SA specification in CHARMY is available, the Charmy2Promela translation feature is used to obtain from the model-based SA specification a formal executable prototype in Promela. The Charmy2Promela translation algorithm has been already presented elsewhere [8] and does not represent the key novelty of this approach.

On the generated Promela code we can use the SPIN standard features to find, for instance, deadlocks or parts of states machines that are unreachable, or to simulate the architecture. SASIM interprets SPIN results (difficult to be deciphered from a non-SPIN specialist) in terms of CHARMY sequence and state machines. The designer, which diagrammatically described the software architecture, can see the simulation in terms of animation of the same diagrams he designed.

SASIM allows the designer to run step-by-step interactive simulations. Figure 2 shows this cooperation. The foreground window proposes to the designer possible events to be run, according to the current simulation status.

During SPIN simulation, the simulation output is read line by line at run time and represented through state machines and scenarios animations in CHARMY. The algorithm is defined so that information not to be graphically shown, are not considered.

Another relevant point to be considered is that operations treated not atomically by the SPIN simulator, can be atomic in CHARMY. This is due to the fact that the exchange of synchronous messages between two processes requires to activate both the sender and the receiver process. Thus, SASIM waits to have the action accomplished before visualizing them; when the sender is ready for sending the message (synchronous message) but the receiver is not still scheduled for receiving them, SASIM only colors the state of the sender. The transition is colored only when the message is actually exchanged.

Summarizing, the simulation engine is the SPIN simulator and SASIM coordinates and orchestrates them providing the required inputs and interpreting the generated outputs.

In each moment the designer can check the computational state of each component since the current state of each component, in its state diagram, is colored. When a message is exchanged between two components, a message is added on the sequence diagram automatically generated by SASIM that is used to show the history of the simulation.
2.1 SASIM Simulation Features

Five different simulation features are supported by SASIM:

**Random simulation:** in this case the designer is a spectator and in case of multiples possibilities one of the possible alternatives is randomly chosen. The choices are performed by the SPIN simulator. The designer can choose to show the simulation step-by-step instead of running the simulation rapidly. The simulation is potentially infinite in case of infinite behaviors.

**Interactive simulation:** at each step the possible actions of the system are highlighted and the designer is asked to choose one of them. This functionality is particularly useful for the designer to understand the system behaviors and to gain confidence on the specified design.

**Simulation guided by a sequence diagram:** the designer can design a sequence diagram representing a desired or an undesired behavior of the system and SASIM tries to reproduce this trace. Differently from the random or interactive simulation, this kind of simulation does not generate a sequence diagram but the one selected to guide the simulation is animated.

![Figure 3. Simulation guided by a sequence diagram](image)

In case it is impossible to reproduce them, the user is warned. This behavior is shown in Figure 3 that shows a simulation guided by a sequence diagram.

It is important to note that the sequence to be simulated must contain every message required by the simulation for having a reproducible trace.

**Simulation and exhaustive verification:** this functionality allows the use of simulation as support to the verification by using the SPIN model checker. In case of a not valid result of the verification, a counter example is generated and it is reproduced directly on the system state diagrams and a sequence showing the trace is generated by SASIM. This feature helps in understanding the output of the SPIN model checker that is in terms of the Promela code that typically is very different from the software architecture design.

**Simulation for model differencing:** this simulation gets as input a sequence previously defined by the designer and it tries to simulate it on a modified version of the architectural model. If it is possible to reproduce the entire sequence, then both the sequence and the model are aligned. Otherwise, SASIM makes use of the verification to look ahead for a behavior who may reactivate the system. The execution path is calculated by means of “trace assertions”. Trace assertions are properties that are checked by SPIN by means of the constructs “trace” and “notrace” directly introduced into the Promela code of the system. By means of these constructs it is possible to check if the system can exchange a sequence of messages in a well defined temporal order. Thus, a new instance of SPIN is run with the new Promela code containing also the notrace annotations that encode the sequence until the first message that is impossible to reproduce. If the result is not valid, then a counter example containing every additional message that must be exchanged in order to have the last wanted message is generated. Thus, the obtained sequence diagram (containing the added messages needed to execute the desired sequence diagram) is proposed to the designer since it represents an adaptation of the sequence in the new version of the system. At this point the simulation returns to be driven by the original sequence until the entire sequence is executed or another message that cannot be executed is found. At the end of the process the added messages are colored with a different color in order to be easily distinguishable.

3 Comparison with Related Work

Figure 4 lists the existing software architecture-level simulation methods, and outlines how SASIM compares with them.

Existing simulation techniques mostly provides random simulations (i.e., the trace to be executed is randomly chosen by the simulator). While relevant, random simulation is not always the most cost-effective simulation technique. SASIM provides simulation guided by the counter example (i.e., guided by the error trail generated by the SPIN model checker), simulation guided by a sequence diagram designed by a designer, and interactive simulations.

SASIM supports the combination of simulation and formal verification (regarded to be an effective mean to increase state-space coverage or to restrict the search space to be explored via simulation [6]), while existing architecture-level verifiers treat simulation and formal verification as two
different and separate activities.

Existing software architecture-level simulation techniques use dedicated, ad-hoc, simulation engines. SASIM, instead, makes use of the SPIN simulator. The cost of making the simulation engine is then avoided. However, some effort is still required, since the architectural model needs to be transformed to be given as input to the simulation process, and the simulator output needs to be deciphered in terms of the architectural model. Two are the main advantages of this choice: on one hand the simulation engine embodies several years of experiences on state storage and optimization; on the other hand the simulation engine is natively integrated with the SPIN model checker so to allow a real and useful integration between simulation and exhaustive verification.

While many tools are available for system-level simulation (see [4] for a relevant list of software tools and languages for simulation), and some tools allow for design level simulation of UML-based models (like Simulink, Rose Real Time, Rhapsody, Artisan Studio, Theseus, vUML) this work explicitly focuses on tools specifically supporting simulation at the software architecture level. It is important to remark that this decision has left out tools that in principle are similar to our proposal, but at different abstraction levels.

4 Conclusions

This paper presented SASIM that is a tool for simulating the behavior of software architectures.

Several simulation features are supported: random simulation, interactive simulation, and simulation guided from both the counter-example generated by SPIN (in case of not valid verification) and from a sequence designed by a user. In addition to these well known simulations, SASIM proposes an additional feature that, making use of the model checker engine, tries to simulate a defined sequence diagram on a version of the system that is new with respect to the one on which the designer defined the sequence.

Instead of building a new simulation engine SASIM makes use of the simulation engine provided by the SPIN model checker. For this end, SASIM contains automatic translation to and from the SPIN input and output, respectively. The result is a simulation engine that is well integrated with CHARMY, the tool used for modeling software architectures.

SASIM provides at the moment only behavioral simulation without taking into account performance, security, etc.. As future work we plan to investigate how to apply the same concepts presented in this paper for supporting other types of simulations.

Acknowledgments

This work is partially supported by the POPEYE project: Peer to Peer Collaborative Working Environments over Mobile Ad-Hoc Networks. Sixth Framework Programme. http://www.ist-popeye.org.

References